# An Open-Source Hardware and Software Platform for Telesurgical Robotics Research

*Release 0.00*

Zihan Chen[1], Anton Deguet[1], Simon DiMaio[2], Gregory Fischer[3] and Peter Kazanzides[1]

June 29, 2013

[1]Johns Hopkins University, Baltimore, MD 21218, USA
[2]Intuitive Surgical, Inc., Sunnyvale, CA, USA
[3]Worcester Polytechnic Institute, Worcester, MA, USA

**Abstract**

We present our work to develop a telerobotics research platform that provides complete access to all levels of control via open-source custom electronics and software. The electronics employs an FPGA to enable a *centralized computation and distributed I/O* architecture in which all control computations are implemented in a familiar development environment (Linux PC) and low-latency I/O is performed over an IEEE-1394a (Firewire) bus at speeds up to 400 Mbits/sec. The mechanical components of the system are provided by the da Vinci Research Kit, which consists of the Master Tele-Manipulators (MTMs), Patient Side Manipulators (PSMs), and stereo console of the first-generation da Vinci surgical robot. This system is currently installed, or planned for installation, at 11 research institutions, with additional installations likely in the future, thereby creating a research community around a common open-source hardware and software platform.

## Contents

## 1 Introduction

The prevalence of open source software continues to increase and to enable research in many fields. Medical Image Computing (MIC) is one example, where packages such as the Visualization Toolkit (VTK), the Insight Toolkit (ITK), 3D Slicer, and many others provide a wealth of algorithms for processing and visualizing medical images. In comparison, there are fewer open source packages for Computer Assisted Intervention (CAI). One reason is that CAI systems employ a diversity of hardware platforms, and these platforms often are either proprietary (closed) commercial systems or custom research systems. Some open source CAI packages have achieved modest success by supporting commercial devices, such as tracking systems, that have open interfaces and are relatively widely used in research systems. Examples include the Image Guided Surgery Toolkit (IGSTK) [3] and our own Surgical Assistant Workstation (SAW) [11, 6].

Within CAI, there are few open platforms for research in medical robotics. We consider specifically the area of telesurgery, which requires a *master* input device, preferably with haptic feedback, and a *slave* (or patient-side) robot with the ability to actuate surgical instruments. Currently, there are several haptic input devices with open interfaces, ranging from low-cost systems such as the Phantom Omni and Novint Falcon, to more costly alternatives. On the slave side, the Raven II research robot [8] was recently disseminated to several research groups via support from the National Science Foundation (NSF) and is available for purchase from Applied Dexterity, Inc. (Seattle, WA). The da Vinci Surgical Robot ® (Intuitive Surgical, Inc., Sunnyvale, CA) can be configured to provide a read-only research interface to both the master and slave manipulators [2]. While useful for some research projects (e.g., skill assessment), the read-only interface does not support projects that require external control of the manipulators (e.g., research in autonomous or semi-autonomous control). Few of the above platforms are open at all levels of control, however. In particular, only the Raven II enables researchers to modify the real-time servo control code, which runs on a Linux PC and communicates with the hardware (e.g., motors and encoders) via a USB interface.

We have previously presented the *cisst* libraries [5, 1, 4], that support both real-time device (robot) control and real-time computer vision, which are necessary components of telesurgical robot systems. The SAW package, which is built on *cisst*, includes components that implement interfaces to many CAI devices, including tracking systems, haptic input devices, and robots. Use of this software, however, is predicated on access to the corresponding hardware. In this paper, we describe an “open source mechatronics” system, consisting of hardware, firmware, and software (see Fig. 1) that is being replicated at multiple institutions.

## 2 System Design

The primary design goal is to provide a system that enables researchers to easily implement new algorithms at any level of control. We therefore did not use an off-the-shelf motor controller because it would not allow modification of the low-level servo control algorithm. We assume that researchers will be familiar with a Linux development environment, preferably with either the RT-Preempt patch or a real-time extension such as Xenomai or RTAI, and therefore focused on a system architecture that enables all software to be implemented in this environment.

We considered several design approaches, which can be categorized based on whether the computation is centralized (e.g., on the PC) or distributed (e.g., high-level control on the PC, low-level control on embedded processors) and whether the I/O is centralized (i.e., all cabling brought back to the PC) or distributed (i.e., cables brought to external boards, possibly inside the robot structure).

The *centralized computation and I/O* architecture was prevalent in the early days of robotics, with multiple joint control boards attached to the computer's parallel bus (e.g., ISA, Q-Bus, or VME). This architecture provides low latency I/O and is still used today, with I/O boards installed in the PCI or PCI-Express bus in a PC. Disadvantages of this architecture include: (1) routing a large bundle of cables between the robot and control PC, which often reduces reliability and performance (e.g., due to noise), and (2) powering off and opening the PC to install the I/O boards, which limits the flexibility of configuration for research.

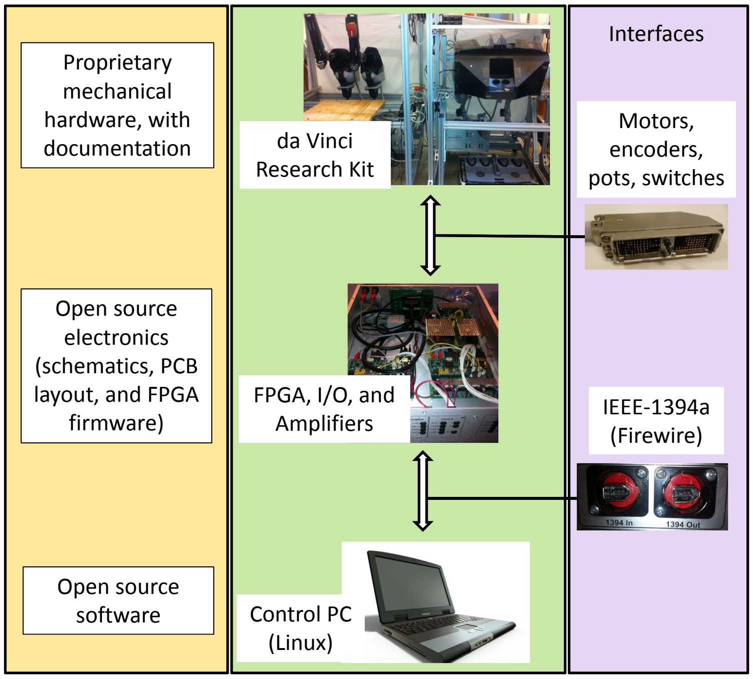
Proprietary mechanical hardware, with documentation
Open source electronics (schematics, PCB layout, and FPGA firmware)
Open source software
da Vinci Research Kit
FPGA, I/O, and Amplifiers
Control PC (Linux)
Interfaces
Motors, encoders, pots, switches
IEEE-1394a (Firewire)
1394 In
1394 Out

Figure 1: Overview of telerobotic research platform: Mechanical hardware provided by da Vinci Research Kit, electronics by open-source IEEE-1394 FPGA board coupled with Quad Linear Amplifier (QLA), and software by open-source *cisst*/SAW package with ROS interfaces.

With the emergence of high-speed serial networks, such as CAN, Ethernet, USB, and IEEE-1394, it became feasible to physically distribute the I/O to external boards. Placing these components inside or near the robot arm allows significant cabling reductions because the thick cables containing multiple wires for motor power and sensor feedback can be replaced by thin network and power cables. But, these systems typically are examples of *distributed computation and I/O* because they perform low-level servo control on the external boards. This is often necessary because even with these high-speed networks, it is difficult to get low enough latency to support servo control at rates of 1 kHz or higher. The disadvantage is that researchers either cannot modify the low-level control (if implemented on a proprietary product) or must learn the idiosyncrasies of embedded programming.

Our approach, however, can be called *centralized computation and distributed I/O* [7]. We achieve this by designing custom electronics that replace the distributed microprocessors with programmable logic; specifically a field-programmable gate array (FPGA). The FPGA provides direct, low-latency, interfaces between the high-speed serial network (IEEE-1394a) and the I/O hardware. In this design, the IEEE-1394 link layer protocol is implemented in the FPGA; see Section 4.3 for further details. This architecture preserves the advantages of reduced cabling, while allowing all software to be implemented on a high-performance com-

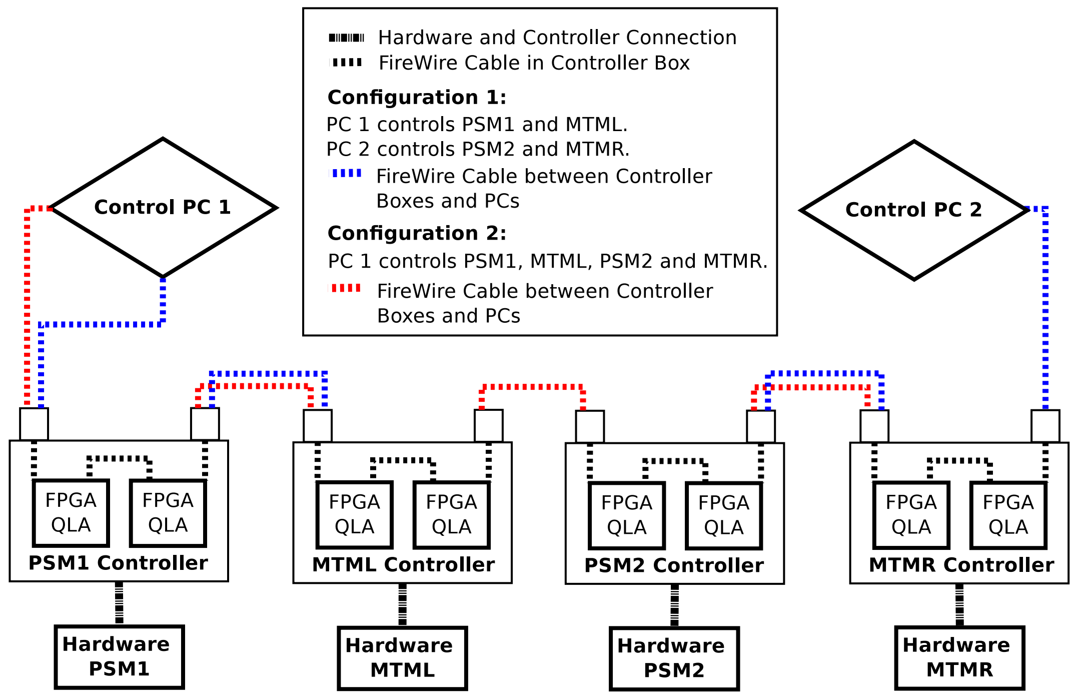
Hardware and Controller Connection
FireWire Cable in Controller Box
Configuration 1:
PC 1 controls PSM1 and MTML.
PC 2 controls PSM2 and MTMR.
FireWire Cable between Controller Boxes and PCs
Configuration 2:
PC 1 controls PSM1, MTML, PSM2 and MTMR.
FireWire Cable between Controller Boxes and PCs
Control PC 1
Control PC 2
FPGA QLA
PSM1 Controller
MTML Controller
PSM2 Controller
MTMR Controller
Hardware PSM1
Hardware MTML
Hardware PSM2
Hardware MTMR

Figure 2: Hardware connection between manipulators, controllers and PCs. Two configurations are shown: (1) one PC controlling all four manipulators, and (2) two PCs controlling PSM1/MTML and PSM2/MTMR, respectively.

puter that contains a familiar software development environment. It also enables flexible reconfiguration; for example, a bimanual teleoperation system (two master robots and two slave robots) can be quickly reconfigured into two independent master/slave systems by disconnecting one IEEE-1394 cable and re-connecting it to a different computer, as shown in Fig. 2.

## 3 Mechanical Hardware

The mechanical hardware is provided by the da Vinci Research Kit (Fig. 3), which consists of the following components of the first-generation da Vinci robot system (often called the da Vinci Classic): two Master TeleManipulators (MTMs), two Patient Side Manipulators (PSMs), a High Resolution Stereo Viewer (HRSV), and a footpedal tray. This kit has been provided by the manufacturer, Intuitive Surgical, Inc., to several research groups. The kit does not include electronics or control software, thereby motivating the development of a common, open-source electronics and software platform for this research community. It also does not include the passive Setup Joints that support the PSMs, the Endoscopic Camera Manipulator (ECM), the stereo endoscope, nor any of the frames or covers.

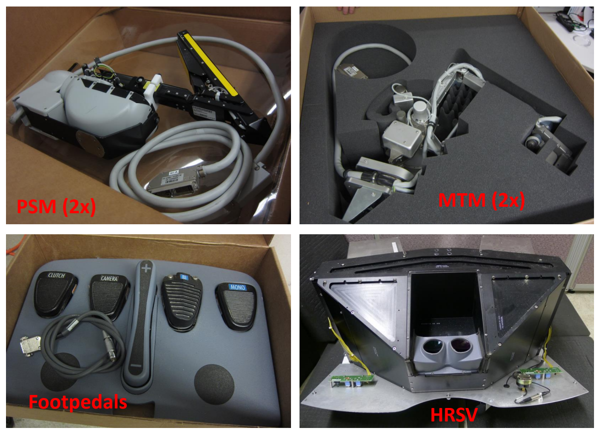

Figure 3: The da Vinci Research Kit

## 4 Electronics

The control electronics is based on two custom boards (Fig. 4): (1) an IEEE-1394 FPGA board, and (2) a Quad Linear Amplifier (QLA). The schematics, firmware, low-level software interface, and documentation are available via a public SVN/Trac repository.

### 4.1 IEEE-1394 FPGA Board

This board contains a Xilinx Spartan-6 XC6SLX45-2 FPGA, configuration PROM, IEEE-1394a physical layer (PHY), two IEEE-1394a 6-pin connectors, a low-speed USB interface, and required power supplies. It contains two 44-pin connectors that provide power and FPGA I/O to a companion board, such as the QLA. It also contains a 16-position rotary switch for board identification.

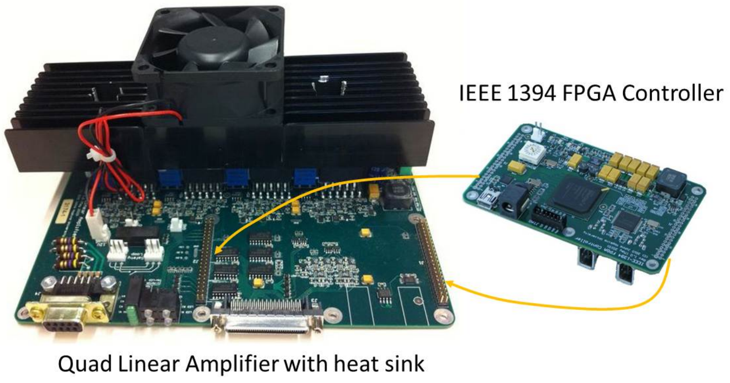
IEEE 1394 FPGA Controller
Quad Linear Amplifier with heat sink

Figure 4: IEEE-1394 FPGA board and Quad Linear Amplifier (QLA)

### 4.2 Quad Linear Amplifier (QLA)

The Quad Linear Amplifier attaches to the IEEE-1394 FPGA board and provides all hardware required to control four DC brush motors, using a bridge linear amplifier design (Fig. 5). Each of the four channels contains the following components:

- One 16-bit digital-to-analog converter (DAC) to enable the FPGA to set the desired motor current.
- Two 16-bit analog-to-digital converters (ADCs) to digitize the measured motor current and an external analog sensor (e.g., potentiometer).
- Differential receivers for one quadrature encoder with A, B, and Z (index) channels; these signals are supplied to the FPGA board, which performs the quadrature decoding.
- Two OPA-549 power operational amplifiers (op amps) to provide bi-directional control of a motor from a single power supply (up to 6.25 Amps at up to 48 Volts).
- Digital inputs for one home and two limit switches; these can also be used as general-purpose inputs.
- One open-collector digital output with high current drive (up to 1 Amp).

The board also contains a software-controlled, normally-open safety relay, which is designed to enable the software to disable the motor power supply, as well as two sensors that measure the heat sink temperature.

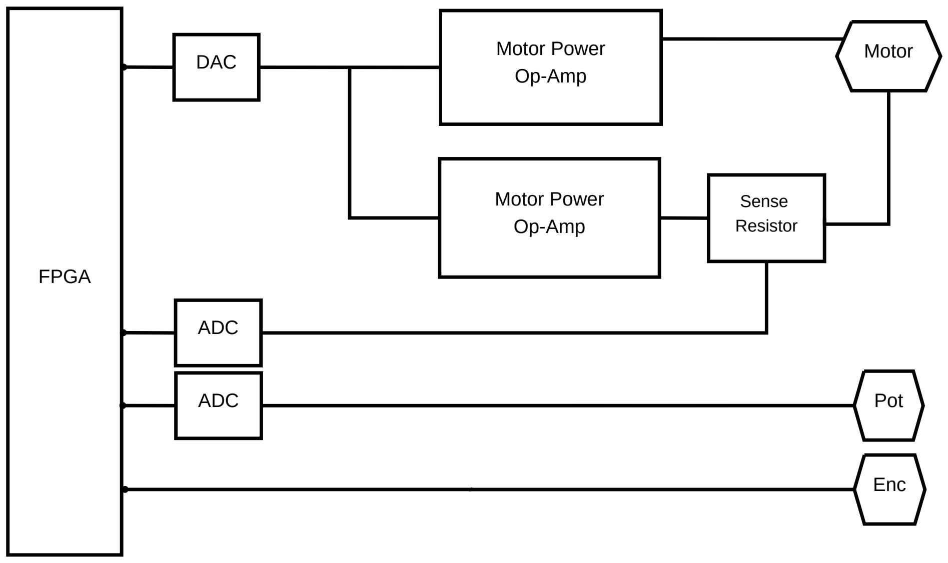

Figure 5: Block diagram of I/O devices

## 4.3 FPGA Firmware

The adoption of an FPGA as the processing chip on the hardware control nodes is driven by the fact that it provides a low-latency interface to the hardware, as well as significant computational power via built-in Digital Signal Processing (DSP) slices. This is crucial for our *centralized computation and distributed I/O* architecture, although one could easily switch to a *distributed computation and I/O* architecture by incorporating a microprocessor core in the FPGA or by using the DSP slices to implement the low-level control algorithms. In general, the FPGA firmware has three major functionalities: (1) responding to read and write requests from the PC via the 1394 bus, (2) interfacing to I/O devices, and (3) hardware-level safety checking.

The IEEE-1394 protocol supports two types of services: isochronous and asynchronous transfers. In isochronous mode, a packet of variable length is broadcast on a specified channel (up to 64 channels) at a guaranteed 8 kHz (125 $\mu$s) bus cycle. It is ideal for applications such as audio or video data transfer that require constant transfer rates but not necessarily data integrity, since there are no acknowledgements. In contrast, asynchronous transfers deliver a packet to a specific address (node id) and require an acknowledgement. If an acknowledgement is missing or invalid, a retry may be issued. We selected asynchronous transfers for our application because they could satisfy our goal of performing servo control at a 1 kHz rate (or better) and because the requirement for acknowledgment packets improves robustness. To save FPGA

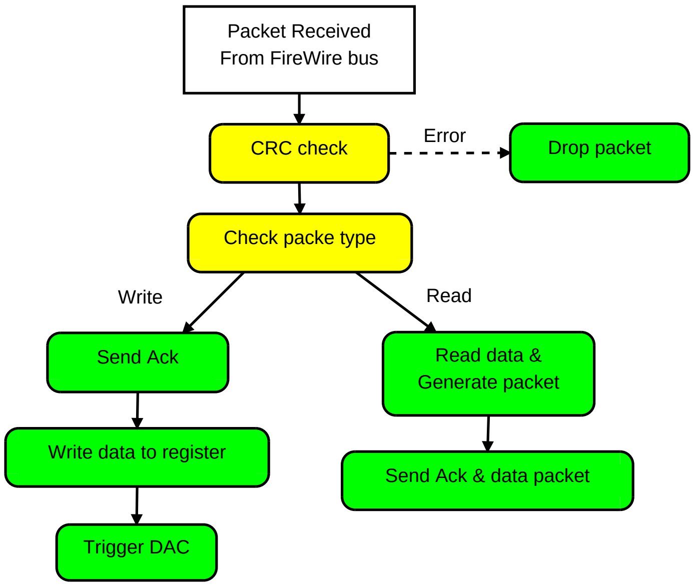

Figure 6: FPGA read and write packet processing

gate resources and to simplify implementation, we implement only a subset of the IEEE-1394 link-layer protocol. Specifically, our FPGA nodes are not capable of serving as bus master, all transfers much be asynchronous quadlet (32-bit) or block (multiple quadlets) read or write transactions. The lack of a bus master implementation is not a serious limitation because we can rely on the IEEE-1394 interface in the PC to fill this role.

Figure 6 summarizes how the FPGA handles read and write transactions. When the FPGA receives a write packet over the IEEE-1394 bus, it does the following four things: (1) checks the incoming packet's Cyclic Redundancy Check (CRC) value and silently drops the packet if the CRC is invalid, (2) generates and sends an acknowledgement packet, (3) decodes the destination device address and data from the packet, and (4) writes data to internal resigters and to the different I/O devices. For example, the desired motor current is shifted out via the Serial Peripheral Interface (SPI) to the DAC. Similarly, to respond to a read request from the PC, the FPGA latches various I/O device data and sends all requested data back to the PC in a single block transfer. To avoid latency, the FPGA ensures that all data that could be requested is available in local registers. For example, because one ADC conversion cycle requires 0.7 $\mu s$, the FPGA firmware continuously requests data conversions and stores the result in a register for future read requests.

In addition to the read and write requests to the devices involved in motor control, the FPGA firmware also supports reading and writing to the configuration PROM that initializes the FPGA. It is therefore possible to update the firmware vie the IEEE-1394 interface, which provides several advantages: (1) no special JTAG programming cable is required, (2) no special programming software is required, and (3) it is much faster than the conventional JTAG programming method (about 20 seconds versus several minutes).

Although the goal of the *centralized computation* architecture is to enable researchers to implement all software on the PC, we decided to implement certain safety features directly in the FPGA. This has two primary advantages: (1) the FPGA is naturally a hardware implementation and thus provides fast response and reliability, and (2) the safety feature is always active, even if the software crashes or the Firewire cable becomes disconnected. The current firmware includes two major safety features: a watchdog timer and a motor current safety check. The watchdog timer provides a range of timeout periods from 1 to 340 ms (setting the period to 0 disables the watchdog). If the watchdog is not refreshed during this period (by writing to the FPGA), it trips and disables all power amplifiers. This safety mechanism is especially useful to turn off the motors in situations where the PC control software exits or communication is lost. The motor current safety module is designed to catch cases where the absolute value of the measured motor current is significantly greater than the commanded motor current, which would indicate a hardware defect.

## 5 Software

Under the principle of *centralized computation and distributed I/O*, all computation including data read/write, servo loop control and high level robot control are implemented on a Linux PC. This section introduces the low-level C++ API that interacts with FPGA controllers via IEEE-1394 bus and the high level robot control software, which also includes interfaces to the Robot Operating System (ROS) [9].

### 5.1 Low-Level Interface

A low-level C++ library is available to allow direct access to the raw I/O data in our customized hardware through the IEEE-1394 bus. This library has no external software dependencies. It uses the `libraw1394` library, which is a standard Linux library for communication over IEEE-1394. Other drivers, such as RT-Fire [12], could be used to obtain hard real-time performance. In prior experiments [10], we discovered that

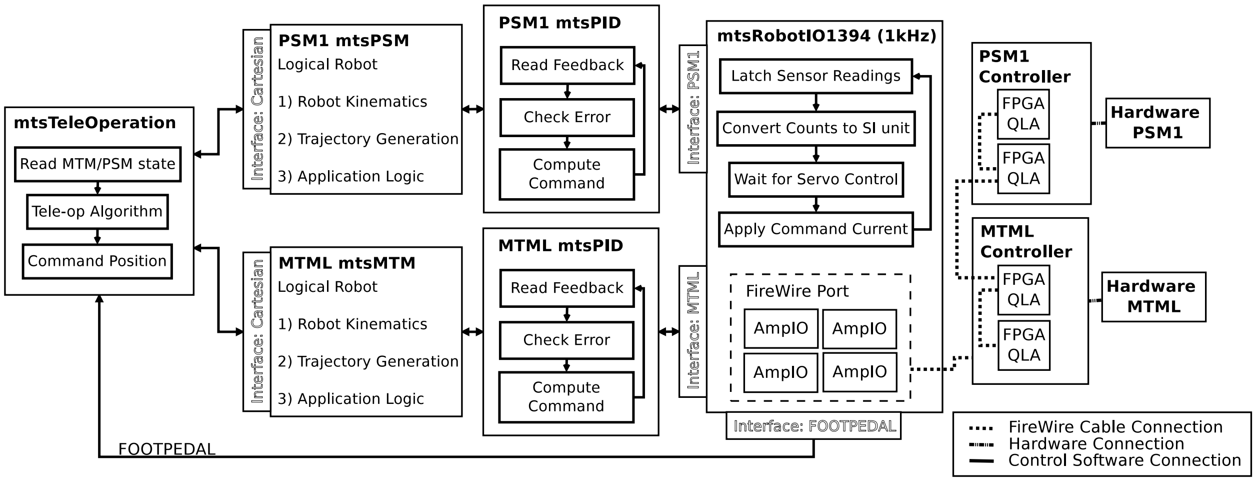

Figure 7: Robot tele-operation control architecture with one MTM and one PSM

most of the latency in IEEE-1394 data transfers appears to be due to overhead on the PC. Specifically, we found that small packets had a latency of about 30-35 $\mu$sec, whereas the latency for large packets was only a little larger. Therefore, we determined that to achieve the best performance, we should combine reads and writes as much as possible. Currently, we combine the read and writes for each individual board, so that for N boards, we have N reads and N writes. Even better performance could be obtained by combining packets further; for example, it is relatively straightforward to combine the N writes into a single broadcast packet.

The API consists of two main classes: a `FirewirePort` class to represent an IEEE-1394 port and an `AmpIO` class to represent one FPGA node on the bus. One Firewire port can contain multiple FPGA nodes. In the real-time control mode, the Firewire port latches all block reads from the FPGA boards into local buffers on the PC, and then applies the new motor currents to the control boards via block writes. The `AmpIO` API provides a set of functions to extract feedback data, such as encoder positions, from the read buffer, and to write data, such as desired motor currents and watchdog timer period, into the write buffer. All data formats in this API are unsigned integers because data are stored as counts (or bits) in FPGA registers. Also, various I/O devices in the controller boards expect different data formats; thus the high level control software calling these API functions is responsible for correctly formatting the data. Besides the basic API, a set of utility programs are provided to print Firewire port node information and talk directly to the FPGA in raw format.

## 5.2 Component-Based Control Architecture

We employ a component-based control architecture, using the open-source *cisst*/SAW libraries available via a public SVN/Trac repository. As shown in Figure 7, in a tele-operation system with one MTM and one PSM, the control system contains control components for I/O level read/write, servo control, logical robot control and a tele-operation control component. In this control architecture, components have well-defined required and provided interfaces and are inter-connected through these interfaces. The *mtsRobotIO1394* component communicates with the hardware directly via the IEEE-1394 bus and provides three interfaces in this example: PSM1 I/O level interface, MTML I/O level interface and an interface for footpedal events. The PSM1/MTML *mtsPID* servo controllers connect to PSM1 and MTML I/O level interfaces, respectively. Also, the *Tele-operation* component connects to the footpedal interface and adjusts its control behavior based on footpedal events. Similarly, the *mtsPSM/mtsMTM* expose the Cartesian level interfaces and require a joint level interface from the *mtsPID* servo loop component.

One challenge for such a component-based approach is data synchronization; this is especially true for servo loop control running at high frequency of one kilohertz. If a separate thread is created for each servo control loop and the I/O component, it is likely that the feedback data used in the servo loop control could be out of sync and potentially affect controller performance. Our solution puts all servo control loops and the I/O component in one single thread while keeping the advantage of a component-based approach.

Two types of logical robot, *mtsPSM* and *mtsMTM*, have been created to handle forward kinematics, inverse kinematics, trajectory generation, robot-specific control (e.g. gripper open angle) and most important high level robot state control. The *Master Manipulator* has 7 active joints plus one passive gripper with two Hall effect sensors. The gripper is a separate input device independent of the MTM's kinematics and needs to be processed separately in this logical robot component. The *Patient-Side Manipulator* also has 7 active joints driven by 7 actuators. Through an adapter, a variety of instruments, such as forceps and scissors, can be installed onto the PSM to perform different tasks during surgery. The instrument is driven by the last 4 actuators with a cable-driven mechanism. If the instrument mounted on the PSM contains a gripper, it also requires specialized processing in *mtsPSM*. Tool adapter and instrument installation require an engagement procedure involving the last 4 actuators in the logical robot component.

### 5.3 ROS Interface

ROS (Robot Operating System) provides a communications layer, which eases the communication between different robot control software processes in one computer or across multiple computers [9]. It also provides a set of libraries and utility tools. To leverage the ROS toolchain and engage developers from the ROS community, we developed an interface that publishes the robot state in ROS messages. This section explains our approach of publishing data as ROS messages (topics).

The control software discussed above is written using the *cisst* library. To publish robot information in ROS messages, synchronized data need to be sampled, converted to ROS message format and then published periodically. The solution is to use an independent *sawROS* library, which is comprised of a set of global data type conversion functions (e.g. *cisst* matrix to ROS geometry_msgs Tranform) and a publisher, running periodically at a fixed rate, to fetch data, convert data and then publish data. By default, the publisher runs at a rate of 20 Hz and can be configured by setting the *da_vinci_update_rate* ROS parameter (up to 100 Hz).

## 6 Conclusions

This paper presented a telerobotics research platform that is based on the da Vinci Research Kit, open-source electronics, and open-source software. This platform is being replicated at several research institutions – currently 11 sites have either acquired or ordered the platform. The user community and collaboration tools are still being formed; at present, there are SVN/Trac repositories (including wikis) for the custom electronics and software. In the near future, we expect to create a mailing list, as well as additional wikis to document other aspects of the platform.

## Acknowledgments

This work was supported by National Science Foundation grants EEC 9731748, EEC 0646678, and MRI 0722943, and by NASA NNX10AD17A. Paul Thienphrapa, Simon Leonard, Kwang Young (Eddie) Lee, and Ravi Gaddipati provided technical assistance at JHU.

## References

[1] A. Deguet, R. Kumar, R. Taylor, and P. Kazanzides. The *cisst* libraries for computer assisted intervention systems. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1465, Sep 2008. 1

[2] S. DiMaio and C. Hasser. The da vinci research interface. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1464, July 2008. 1

[3] K. Gary, L. Ibanez, S. Aylward, D. Gobbi, M. Blake, and K. Cleary. IGSTK: an open source software toolkit for image-guided surgery. *IEEE Computer*, 39(4):46–53, Apr 2006. 1

[4] M. Y. Jung, A. Deguet, and P. Kazanzides. A component-based architecture for flexible integration of robotic systems. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 6107–6112, 2010. 1

[5] A. Kapoor, A. Deguet, and P. Kazanzides. Software components and frameworks for medical robot control. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3813–3818, May 2006. 1

[6] P. Kazanzides, S. DiMaio, A. Deguet, B. Vagvolgyi, M. Balicki, C. Schneider, R. Kumar, A. Jog, B. Itkowitz, C. Hasser, and R. Taylor. The Surgical Assistant Workstation (SAW) in minimally-invasive surgery and microsurgery. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal, Jun 2010. 1

[7] P. Kazanzides and P. Thienphrapa. Centralized processing and distributed I/O for robot control. In *Technologies for Practical Robot Applications (TePRA)*, pages 84–88, Woburn, MA, Nov 2008. 2

[8] H. King, S. N. Kosari, B. Hannaford, and J. Ma. Kinematic analysis of the Raven-II(tm) research surgical robot platform. Technical Report 2012-0006, Department of Electrical Engineering, University of Washington, Jun 2012. 1

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009. 5, 5.3

[10] P. Thienphrapa and P. Kazanzides. A scalable system for real-time control of dexterous surgical robots. In *Technologies for Practical Robot Applications (TePRA)*, pages 16–22, Nov 2009. 5.1

[11] B. Vagvolgyi, S. DiMaio, A. Deguet, P. Kazanzides, R. Kumar, C. Hasser, and R. Taylor. The Surgical Assistant Workstation: a software framework for telesurgical robotics research. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, Midas Journal: http://hdl.handle.net/10380/1466, Sep 2008. 1

[12] Y. Zhang, B. Orlic, P. Visser, and J. Broenink. Hard real-time networking on FireWire. In *RT Linux Workshop*, Nov 2005. 5.1